

Dokumentácia projektu

Interpret jazyka IFJ14

Varianta a/1/II

Rozšírenia MINUS, BASE, REPEAT, FUNEXP, ELSEIF, BOOLOP

13. decembra 2014

Vedúci tímu 002:	Dávid Mikuš (xmikus15)	20%
Členovia:	Peter Hostačný (xhosta03)	20%
	Tomáš Kello (xkello00)	20%
	Adam Lučanský (xlucan01)	20%
	Michaela Lukášová (xlukas09)	20%

Obsah

1	Úvod	1
2	Riešenie projektu	1
2.1	Lexikálna analýza	1
2.2	Syntaktická a sémantická analýza	1
2.3	Inštrukcie a interpret	1
3	Algoritmy a datové štruktúry	2
3.1	Quicksort	2
3.2	Knuth-Morris-Pratt	2
3.3	Tabuľka s rozptýlenými položkami	2
3.4	Vector - pole s neobmedzenou kapacitou	3
3.5	Reťazec	3
4	Práca v tíme	3
A	Metriky kódu	3
B	Konečný automat lexikálneho analyzátora	4
C	LL Gramatika	5
D	Precedenčná tabuľka	6

Notes

■	Doplniť používanie Gitu, rozdelenie práce	3
■	Aktualizovať metriky po dopísaní projektu	3

1 Úvod

Cieľom projektu bolo vytvoriť funkčný interpret jazyka IFJ14, ktorý má veľmi podobnú syntax ako jazyk Pascal.

Táto dokumentácia pojednáva o základných princípoch použitých pri riešení.

2 Riešenie projektu

V tejto kapitole budú rozpísané jednotlivé časti s dôrazom na vlastné riešenie.

Vstupom interpretu je zdrojový súbor, ktorý je priamo vykonaný, namiesto generovania výstupného spustiteľného súboru.

2.1 Lexikálna analýza

Vstupný súbor reprezentovaný znakmi je nutné rozdeliť na „tokeny“. O túto činnosť sa stará tzv. „scanner“, ktorý je reprezentovaný konečným automatom (viz prílohu **B**). Dosiahnutie koncového stavu značí úspešný nález tokenu.

Konverzie medzi reťazcami a číslami sa vykonávajú ihneď po vrátení platného tokenu. Konverzie v rámci escape sekvencie v reťazcovom literáli sa vykonávajú priamo v automate pri prechode medzi stavmi.

2.2 Syntaktická a sémantická analýza

Syntaktická analýza bola riešená zadaním cez rekurzívny zostup podľa gramatiky v prílohe **C**. Inštrukcie pre interpret sú generované okamžite na inštrukčnú pásku, bez dodatočného medzivýstupu.

2.3 Inštrukcie a interpret

Ako najzaujímavejšia časť celého interpretu je generovanie a vykonávanie inštrukcií z inštrukčnej pásky vzhľadom k voľnosti riešenia tejto časti.

Interpret pri svojej činnosti pracuje so zásobníkom (o neobmedzenej veľkosti) a inštrukčnou páskou, ktorá obsahuje inštrukcie v trojadresnom kóde. Každý operand obsahuje informáciu o tom

- kde sa nachádzajú údaje (ďalej len lokalita):
 - globálne
 - lokálne (Base Pointer + Offset)
 - konštanta (okamžitá hodnota)
- akého sú dátového typu (integer, real, boolean, string)

Keďže interpret spracováva inštrukcie na úrovni C funkcií, pri každej inštrukcii sa najprv musí rozhodnúť akého je dátového typu a kde sa nachádzajú údaje pre každý operand, čo stojí veľké množstvo času pri samotnom behu v porovnaní s efektívnym výpočtom. Rozhodli sme sa teda vyriešiť tento problém elegantným spôsobom, a to zatiaľ parser o jednu činnosť navyše - generovanie konkrétnych inštrukcií pre interpret, obsahujúce v sebe zakódované lokality a dátové typy.

Vzhľadom k charakteru inštrukcií sme pomocou napísaného skriptu v jazyku Ruby vygenerovali 358 inštrukcií, ktoré priamo v mene obsahujú lokalitu a typy operandov. Inštrukcie sú uložené v poli ukazovateľov na funkcie (inštrukcie). Pri vykonávaní programu sa teda volá C funkcia pomocou ukazovateľa ktorý v sebe obsahuje.

Interpret beží v nekonečnom cykle, pričom končí v momente keď narazí na inštrukciu HALT.

3 Algoritmy a datové štruktúry

3.1 Quicksort

Funkcia na zoradenia prvkov v poli. Ako prvé si zadefinujeme indexy okrajových hodnôt medzi ktorými sa nachádzajú prvky na zoradenie spolu so strednou hodnotou indexu ktorý určíme ako pivot.

Následne posúvame ukazovatele z pravej a ľavej strany smerom k pivotu, v prípade že nevyhovujú postupnosti z ľavej strany od najnižšej hodnoty, vtedy tieto dve okrajové hodnoty prehodíme, až kým neprejdeme celým polom. Takýmto spôsobom presne definujeme pozíciu pivota v poli. Všetky prvky na ľavej strane od neho sú menšie a na pravej väčšie ako jeho hodnota, avšak nie sú v správnom poradí. Preto sa rekurzívne volá rovnaká funkcia, avšak už len nad pravou a ľavou stranou ktoré sú menšie. Takýmto spôsobom prejdeme celé pole ktoré sa stane usporiadaným.

3.2 Knuth-Morris-Pratt

Vyhľadávanie podreťazca v reťazci je vo vstavanej funkcii riešené algoritmom Knuth-Morris-Pratt, ktorý je založený na vytvorení masky hľadaného textu. Masku určuje program ako sa má chovať v prípade, že sa písmeno hľadaného textu nezhoduje.

1	2	3	4	5	6	7	8
A	T	T	A	T	A	C	A
0	0	0	1	2	1	0	2

Tabuľka 1: Príklad masky algoritmu KMP

Maska je pole o dĺžke hľadaného textu a obsahuje celočíselné nezáporne čísla. Ku každému písmenu je priradené číslo ktoré udáva index, kam sa má program vrátiť v prípade, že nenastane zhoda na danej pozícii. Príklad takej masky pre slovo „ATTATACA“ sa nachádza v tabuľke 1.

Význam tohto algoritmu vidíme v prípade, že uprostred slova sa nachádza rovnaká sekvencia ako na začiatku prehľadávaného textu. To znamená, že ak vo vstupnom texte nastane nezhoda na indexe 6, avšak predchádzajúcich 5 znakov už bolo skontrolovaných, podľa masky vieme vyčítať posun na index 2. Inak povedané, písmena s indexmi 4..5 sú zhodné s 1..2 a preto ich netreba kontrolovať znovu, ale môžeme pokračovať na indexe 3. Ak ani na treťom indexe neuspelo porovnanie, vraciame sa na začiatok, t.j. index „0“. Takto pokračujeme až kým nedosiahneme koniec vstupu alebo nenájdeme celý reťazec vo vstupnom texte.

3.3 Tabuľka s rozptýlenými položkami

Táto dátová štruktúra bola použitá na uschovávanie lokálnych symbolov v danej funkcii. Jej výhodou je rýchlosť hľadania symbolov. Základom celej tabuľky je pole ukazovateľov na jednosmerne viazané zoznamy. Každá položka v zozname obsahuje ukazovateľ na následníka a samotný symbol.

Na základe symbolu sa získa index do pola ukazovateľov, v ktorom sa ďalej sekvenčne prechádza, až dokým sa nenájde požadovaná položka, alebo nie je dosiahnutý koniec zoznamu.

3.4 Vector - pole s neobmedzenou kapacitou

Pre účely interpretu bolo nevyhnutné vytvoriť vlastný datový typ pre prácu s polami rôznych dĺžok, ktoré bežné polia neumožňujú, definovali sme si typ `Vector`, inšpirovaný vektorom z jazyka C++.

`Vector` je štruktúra, ktorá si nesie informácie o alokovanom a použitom počte položiek spolu so samotným ukazovateľom na začiatok alokovaného priestoru. Pre prácu s vektorom sú definované špeciálne funkcie, ktoré v prípade zaplnenia realokujú potrebný priestor na zápis položky. Realokuje sa vždy na dvojnásobok predchádzajúcej veľkosti.

3.5 Reťazec

Podobne ako dátový typ `Vector` bolo nutné vytvoriť aj `String`, ktorý funguje na prakticky tom istom základe realokácie v prípade potreby, avšak s odlišnými operáciami (konkatenácia, dĺžka).

4 Práca v tíme

Vývoj na projekte sa započal dňa 25. augusta prvým commitom. Pred samotným zadaním projektu sa započala práca na základných štruktúrach a prekladovom systéme.

Doplniť používanie Gitu, rozdelenie práce

A Metriky kódu

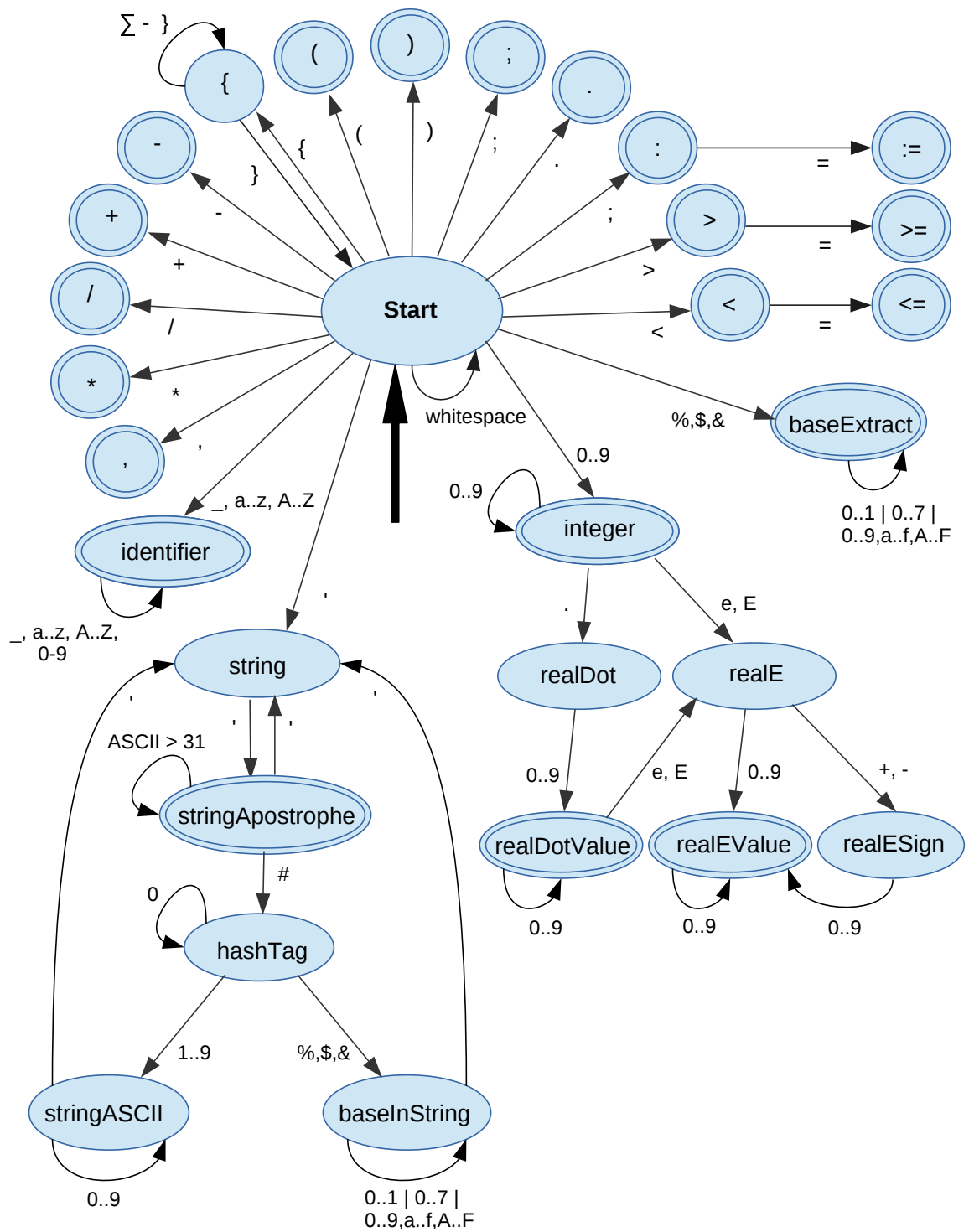
Počet riadkov zdrojového textu: 12689 riadkov

Veľkosť statických dát: 6147B

Veľkosť spustiteľného programu: 13294B

Aktualizovať metriky po dopísaní projektu

B Konečný automat lexikálneho analyzátor



C LL Gramatika

- 1: $PROGRAM \rightarrow VAR_DECLR\ FUNC\ COMPOUND_STMT .$
- 2: $VAR_DECLR \rightarrow \text{var } VAR_DEF$
- 3: $VAR_DECLR \rightarrow \epsilon$
- 4: $VAR_DEF \rightarrow \text{id} : \text{type} ; VAR_DEFN$
- 5: $VAR_DEFN \rightarrow \text{id} : \text{type} ; VAR_DEFN$
- 6: $VAR_DEFN \rightarrow \epsilon$

- 7: $FUNC \rightarrow \text{function id } PARAM_DEF_LIST : \text{type} ; FORWARD\ FUNC$
- 8: $FUNC \rightarrow \epsilon$
- 9: $FORWARD \rightarrow \text{forward} ;$
- 10: $FORWARD \rightarrow VAR_DECLR\ COMPOUND_SEMICOLON_STMT$

- 11: $PARAM_DEF_LIST \rightarrow (PARAMS_DEF)$
- 12: $PARAMS_DEF \rightarrow \text{id} : \text{type } PARAMS_DEF_N$
- 13: $PARAMS_DEF \rightarrow \epsilon$
- 14: $PARAMS_DEF_N \rightarrow ; \text{id} : \text{type } PARAMS_DEF_N$
- 15: $PARAMS_DEF_N \rightarrow \epsilon$

- 16: $TERM_LIST \rightarrow (TERMS)$
- 17: $TERMS \rightarrow \text{term } TERMS_N$
- 18: $TERMS_N \rightarrow , \text{term } TERMS_N$
- 19: $TERMS_N \rightarrow \epsilon$

- 20: $COMPOUND_STMT \rightarrow \text{begin } STMT_E \text{ end}$
- 21: $COMPOUND_SEMICOLON_STMT \rightarrow COMPOUND_STMT ;$
- 22: $STMT_LIST \rightarrow \epsilon$
- 23: $STMT_LIST \rightarrow ; STMT\ STMT_LIST$
- 24: $STMT_E \rightarrow STMT\ STMT_LIST$
- 25: $STMT_E \rightarrow \epsilon$
- 26: $STMT \rightarrow \text{id} := EXPR$
- 27: $STMT \rightarrow \text{if } EXPR \text{ then } COMPOUND_STMT\ IF_N$
- 28: $STMT \rightarrow \text{while } EXPR \text{ do } COMPOUND_STMT$
- 29: $STMT \rightarrow \text{repeat } STMT\ STMT_LIST \text{ until } EXPR$
- 30: $STMT \rightarrow COMPOUND_STMT$
- 31: $STMT \rightarrow \text{readln } (\text{id})$
- 32: $STMT \rightarrow \text{write } TERM_LIST$

- 33: $IF_N \rightarrow \text{else } COMPOUND_STMT$
- 34: $IF_N \rightarrow \epsilon$

D Precedenčná tabuľka

	⁻¹	not	*	/	and	+	-	or	xor	<	>	<=	>=	=	<>	()	f	,	\$	var
⁻¹	<i>H</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
not	<i>R</i>	<i>H</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
*	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
/	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
and	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
+	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
-	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
or	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
xor	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
<	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
<=	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
>=	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
=	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
<>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>S</i>
(<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>H</i>	<i>S</i>	<i>H</i>	<i>E</i>	<i>S</i>
)	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>E</i>	<i>R</i>	<i>E</i>	<i>R</i>	<i>R</i>	<i>E</i>
f	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>H</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>	<i>E</i>
,	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>H</i>	<i>S</i>	<i>H</i>	<i>E</i>	<i>S</i>
\$	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>E</i>	<i>S</i>	<i>E</i>	<i>E</i>	<i>S</i>
var	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>E</i>	<i>R</i>	<i>E</i>	<i>R</i>	<i>R</i>	<i>E</i>

R – reduce, H – handle, S – shift, E – error

¹Unárne mínus